INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2003; **1**:1–21 *Prepared using* nmeauth.cls *[Version: 2000/01/19 v2.0]*

A comparison of algorithms for modal analysis in the absence of a sparse direct method

Peter Arbenz[1] and Richard B. Lehoucq[2]

[1] *Swiss Federal Institute of Technology (ETH), Institute of Scientific Computing, CH-8092 Zurich, Switzerland (arbenz@inf.ethz.ch)*
[2] *Sandia National Laboratories, Computational Mathematics & Algorithms, MS 1110, P.O.Box 5800, Albuquerque, NM 87185-1110 (rblehou@sandia.gov)*[3].

SUMMARY

The goal of our report is to compare a number of algorithms for computing a large number of eigenvectors of the generalized eigenvalue problem arising from a modal analysis of elastic structures using preconditioned iterative methods. The shift-invert Lanczos algorithm has emerged as the workhorse for the solution of this generalized eigenvalue problem; however a sparse direct method is required for the resulting set of linear equations. Instead, our report considers the use of preconditioned iterative method; in particular, we employ a scalable algebraic multigrid (AMG) preconditioner. We present a review of available preconditioned eigensolvers and a numerical comparison on two problems.

KEY WORDS: Eigenvalues, large sparse symmetric eigenvalue problems, modal analysis, preconditioning, multigrid

1. Introduction

The goal of our report is to compare a number of algorithms for computing a large number of eigenvectors of generalized eigenvalue problem

$$\mathbf{K}\mathbf{x} = \lambda\mathbf{M}\mathbf{x}, \qquad \mathbf{K}, \mathbf{M} \in \mathbb{R}^{n\times n}, \tag{1}$$

using preconditioned iterative methods. The matrices $\mathbf{K}$ and $\mathbf{M}$ are large, sparse and symmetric positive semi-definite and arise in a modal analysis of elastic structures. The order n is typically of order $10^5 - 10^6$ and several hundred to thousand eigenvectors are often required as the frequency range of interest for the modal analysis increases. The current state of the art is

*Correspondence to: Rich Lehoucq, Sandia National Laboratories, Computational Mathematics & Algorithms, MS 1110, P.O.Box 5800, Albuquerque, NM 87185-1110

Contract/grant sponsor: The work of Dr. Arbenz was in part supported by the CSRI, Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy; contract/grant number: DE-AC04-94AL85000

Received March 12, 2003
Revised March 12, 2003

to use a block Lanczos [14] code; however a sparse direct method is needed for the shift-invert transformation $(\mathbf{K}-\sigma\mathbf{M})^{-1}\mathbf{M}$ employed. This algorithm is commercially available and is incorporated in the MSC.Nastran finite element library.

Unfortunately, the cost of the above partial eigensolution may dramatically increase as the size of the problem and/or the frequency range for the analysis increases. High modal density (close spacing of eigenvalues) also contributes to the cost. The three major costs associated with a shift-invert block Lanczos code are

1. factoring $\mathbf{K}-\sigma\mathbf{M}$;
2. solving linear systems with the above factor;
3. the cost (and storage) of maintaining orthogonality of hundreds, perhaps, thousands of Lanczos vectors.

The efficient implementation of such an approach is feasible as long as the problem size n is not so large so that the above three costs are prohibitive. We conclude our discussion with three important points concerning the cost and effectiveness of a shift-invert block Lanczos code. The first point to make is that the cost associated with 1 and 2 grows nearly quadratically with increasing dimension n and so the approach is not scalable. Loosely, by scalable we mean that the computational cost of a solution strategy is proportional to the problem size. The second point is that when 1 and 2 can be carried out, Lanczos quickly produces accurate approximations to the eigenvalues and eigenvectors near σ and so a sequence of factorizations with shifts $\sigma = \sigma_1, \ldots, \sigma_p$ are typically computed. The third point is that the cost of maintaining orthogonality of the Lanczos vectors is minimized by a careful implementation of the Lanczos three term recurrence. The reader is referred to [14] for further details and information on a state-of-the-art block Lanczos implementation for problems in structural dynamics.

What if a sparse direct method is no longer a viable option because n is so large that the memory requirements (and associated I/O demands) for a sparse direct solver prove insurmountable? Then we must consider other approaches to computing a large number of eigenvectors. Alternatives that we will consider in this report are the following:

1. replace the sparse direct method with a scalable preconditioned iterative method within the Lanczos algorithm;
2. replace the Lanczos algorithm with a scalable preconditioned eigenvalue algorithm (that perhaps better utilizes preconditioned iterative methods).

The former approach is not new and neither are algorithms for the latter alternative. What we propose is a comparison of several of these alternatives on a representative set of problems.

The goal of our report will be to investigate how these methods perform if quite a large number, say a few hundred eigenvalues and eigenvectors are to be computed when an algebraic multigrid (AMG) preconditioner is used. To the best of our knowledge, there is no comparable study. The recently published paper [6] compares fewer methods and uses an incomplete Cholesky (IC) factorization preconditioner. An IC preconditioner, in contrast to an AMG preconditioner, does not scale with respect to the problem size, specifically, the mesh (see [37] for a recent review of AMG methods). We also remark that the commercial finite element analysis program ANSYS now provides an AMG based solver (the algorithm implemented is an enhanced version of the smoothed aggregation scheme introduced in [38]). The reader is referred to [29] for details and a performance comparison of the AMG solver. Therefore,

our comparison provides an indication of the cost of solving extremely large-scale eigenvalue problems for numerous eigenvalues and eigenvectors.

The alternative algorithms we'll discuss can be broadly classified into two groups. The first group are gradient based schemes that attempt to minimize the Rayleigh-quotient. These include Lanczos and conjugate gradient schemes. The second group are schemes that determine stationary points of the Rayleigh quotient via a Newton iteration. This includes Davidson based methods such as the Jacobi-Davidson algorithm.

All of the algorithms we compare, except for the Lanczos algorithm, only require a single application of the preconditioner per iteration as an approximation to the action of $\mathbf{K}^{-1}$ on a vector. This single application of a preconditioner is in direct analogy to the use of a preconditioned iterative method (for instance, the preconditioned conjugate gradient iteration) for the numerical solution of a PDE with $\mathbf{K}$. In contrast, a Lanczos iteration for (1) requires an *inner iteration* that typically results in several applications of a preconditioner.

We report on numerical experiments that we performed by means of realistic eigenvalue problems stemming from finite element discretizations of elastodynamic problems of structural dynamics and vibrations. Although our problems are of most interest for structural analysts we believe that our results are applicable to problems in other domains such as computational chemistry.

The paper is organized as follows. In sections 2–4 we summarize the three types of algorithms that we are going to investigate: Rayleigh quotient minimization algorithms, inexact Newton iterations for determining the stationary points of the Rayleigh quotient, and the Lanczos iteration. We also provide pseudocode for the implementations that we tested. We hope these prove useful to other researchers (and authors of the algorithms). Sections 5 presents and discuss our numerical experiments.

2. Algorithms I: Rayleigh quotient minimization

Let the eigenvalues of problem (1) be arranged in ascending order,

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n, \tag{2}$$

and let $\mathbf{K}\mathbf{u}_j = \lambda_j \mathbf{M}\mathbf{u}_j$ where the eigenvectors $\mathbf{u}_j$ are assumed to be $\mathbf{M}$-orthonormalized, $\langle \mathbf{u}_i, \mathbf{u}_j \rangle \equiv \mathbf{u}_i^T \mathbf{M}\mathbf{u}_j = \delta_{ij}$. The algorithms that we are considering in this section are designed to exploit the characterization of the eigenvalues of (1) as successive minima of the Rayleigh quotient

$$\rho(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{K}\mathbf{x}}{\mathbf{x}^T\mathbf{M}\mathbf{x}} = \frac{\langle \mathbf{M}^{-1}\mathbf{K}\mathbf{x}, \mathbf{x}\rangle}{\langle \mathbf{x}, \mathbf{x}\rangle}, \qquad \mathbf{x} \neq \mathbf{0} \tag{3}$$

of the matrix pencil $(\mathbf{K}, \mathbf{M})$,

$$\lambda_p = \min_{\substack{\mathbf{x}^T\mathbf{M}\mathbf{u}_j = 0 \\ j=1,\ldots,p-1}} \rho(\mathbf{x}), \qquad p = 1, 2, \ldots, n. \tag{4}$$

Algorithm 2.1 exploits this characterization and provides a skeleton for the algorithms to be discussed in this section. In this basic algorithm, successive eigenpairs are computed by Rayleigh quotient minimization in step (3) and then deflated by restriction in step (4). Of course, in an actual implementation, in step (3) of Algorithm 2.1 the eigenvector $\mathbf{u}_j$ is

ALGORITHM 2.1: **Basic algorithm**

(1) $\mathbf{Q}_0 := []$.
(2) for $j = 1, \ldots, p$ do
(3) *Compute a vector* $\mathbf{u}_j$ *that minimizes the Rayleigh quotient* (3) *in* $\{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{Q}_{j-1}^T \mathbf{M}\mathbf{x} = 0\}$.
(4) $\mathbf{Q}_j := [\mathbf{Q}_{j-1}, \mathbf{u}_j]$.
(5) end for

computed only approximatively. We accept $\mathbf{u}_j$ as an approximation to an eigenvector if the *residual* $\mathbf{r}(\mathbf{u}_j) := (\mathbf{K} - \rho(\mathbf{u}_j)\mathbf{M})\mathbf{u}_j$ is sufficiently small, i.e., if $\|\mathbf{r}(\mathbf{u}_j)\| \leq \varepsilon$ for some given positive value ε. The matrix $\mathbf{Q}_j$ is augmented with $\mathbf{u}_j$ in step (4) when the associated residual satisfies the acceptance test.

2.1. The deflation-accelerated conjugate gradient method

Hestenes and Karush [15] were the first to propose an algorithm to compute the minimum eigenvalue of a symmetric eigenvalue problem by the steepest descent method. Let $\mathbf{x}_k$ be an approximate eigenvector. Then, the next iterate is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \tag{5}$$

where the search direction $\mathbf{p}_k$ is proportional to the negative gradient of the Rayleigh quotient

$$\mathbf{g}(\mathbf{x}) := \mathbf{grad}\, \rho(\mathbf{x}) = \frac{2}{\mathbf{x}^T\mathbf{M}\mathbf{x}} (\mathbf{K} - \rho(\mathbf{x})\mathbf{M})\, \mathbf{x} =: \frac{2}{\mathbf{x}^T\mathbf{M}\mathbf{x}}\, \mathbf{r}(\mathbf{x}) \tag{6}$$

at $\mathbf{x}_k$. The scalar α_k in (5) is chosen so that $\rho(\mathbf{x}_{k+1})$ is minimal. This requires the numerical solution of an order two eigenvalue problem.

Bradbury and Fletcher [8] introduced a conjugate gradient-type algorithm to minimize the Rayleigh quotient. The search direction $\mathbf{p}_k$ in (5) is set to be a linear combination of negative gradient and previous search direction $\mathbf{p}_k = -\mathbf{g}(\mathbf{x}_k) + \beta_k \mathbf{p}_{k-1}$ so that consecutive search directions are conjugate or $\mathbf{K}$-orthogonal. This implies that

$$\beta_k = \mathbf{p}_{k-1}^T \mathbf{K}\mathbf{h}_k / \mathbf{p}_{k-1}^T \mathbf{K}\mathbf{p}_{k-1}, \tag{7}$$

where $\mathbf{h}_k = \mathbf{N}^{-1}\mathbf{g}_k = \mathbf{N}^{-1}\mathbf{g}(\mathbf{x}_k)$ is the *preconditioned* residual at $\mathbf{x}_k$.

In practice, either the Bradbury-Fletcher formula

$$\beta_k = \mathbf{h}_k^T \mathbf{g}_k / \mathbf{h}_{k-1}^T \mathbf{g}_{k-1}. \tag{8}$$

or the one by Polak [28] is employed

$$\beta_k = \mathbf{g}_k^T (\mathbf{h}_k - \mathbf{h}_{k-1}) / \mathbf{g}_{k-1}^T \mathbf{h}_{k-1}. \tag{9}$$

The formulas (7)-(9) are equivalent in the context of linear equations. Computing eigenvalues is however a nonlinear problem. Feng and Owen [10] have executed a comparison and found that strategies (8) and (9) have similar convergence properties and are to be preferred over (7).

ALGORITHM 2.2: **Deflation-accelerated conjugate gradient algorithm (DACG)**

(1) *Choose random vector* $\mathbf{x}_0$, $\mathbf{x}_0^T\mathbf{M}\mathbf{x}_0 = 1$, *that satisfies* $\mathbf{Q}_{j-1}^T\mathbf{M}\mathbf{x}_0 = \mathbf{0}$.
(2) $\mathbf{g}_0 := \mathbf{K}\mathbf{x}_0 - \rho_0\,\mathbf{M}\mathbf{x}_0$ *where* $\rho_0 = \mathbf{x}_0^T\mathbf{K}\mathbf{x}_0$.
(3) *Set* $k := 0$; $\quad v := 1$.
(4) while $\|\mathbf{g}_k\|_2 > tol$ do
(5) *Solve the preconditioned linear system* $\mathbf{N}\mathbf{h}_k = \mathbf{g}_k$
(6) $z := \mathbf{h}_k^T\mathbf{g}_k$; $\quad \beta_k := z/v$; $\quad v := z$.
(7) If $k = 0$ then $\mathbf{p}_k := -\mathbf{h}_k$ else $\mathbf{p}_k := -\mathbf{h}_k + \beta_k\mathbf{p}_{k-1}$ end if.
(8) $\mathbf{p}_k := \mathbf{p}_k - \mathbf{Q}_{j-1}(\mathbf{Q}_{j-1}^T\mathbf{M}\mathbf{p}_k)$. $\quad \mathbf{p}_k := \mathbf{p}_k/\|\mathbf{p}_k\|_M$.
(9) *Let* $(\vartheta_1, \mathbf{d}_1)$ *be the smaller eigenpair of the* 2×2 *eigenvalue problem* $([\mathbf{x}_k, \mathbf{p}_k]^T\mathbf{K}[\mathbf{x}_k, \mathbf{p}_k])\,\mathbf{d} = \vartheta\,([\mathbf{x}_k, \mathbf{p}_k]^T\mathbf{M}[\mathbf{x}_k, \mathbf{p}_k])\,\mathbf{d}$.
(10) *Set* $\rho_{k+1} := \vartheta_1$ *and* $\mathbf{x}_{k+1} := [\mathbf{x}_k, \mathbf{p}_k]\mathbf{d}_1$.
(11) $k := k + 1$.
(12) $\mathbf{g}_k := \mathbf{K}\mathbf{x}_k - \rho_k\,\mathbf{M}\mathbf{x}_k$.
(13) end while
(14) $\mathbf{u}_j = \mathbf{x}_k$.

The Bradbury-Fletcher algorithm embedded in our basic Algorithm 2.1 yields the deflation-accelerated conjugate gradient (DACG) algorithm. We have adopted this name from a long series of papers by Bergamaschi, Gambolati and coworkers [27, 12, 4, 5]. See the the papers [31, 32] by Schwarz for proposed variants of a DACG algorithm.

Algorithm 2.2 gives a listing of DACG. Step (1) initializes the random vector $\mathbf{x}_0$. In step (6), the scalar β_k is computed according to (8). Step (8) orthogonalizes the current search direction $\mathbf{p}_k$ with the column span of $\mathbf{Q}_{j-1}$ that contains the eigenvector approximations.

We now discuss implementation issues associated with Algorithm 2.2. We do not store the matrix $\mathbf{M}\mathbf{Q}_j$ because of memory considerations. In order to maintain the $\mathbf{M}$-orthogonality of the columns of $\mathbf{Q}_j$ to machine precision, the classical Gram-Schmidt algorithm with iterative refinement [16, 7] is employed.

Auxiliary vectors are used for storing $\mathbf{K}\mathbf{x}_k$, $\mathbf{M}\mathbf{x}_k$, $\mathbf{K}\mathbf{p}_k$, and $\mathbf{M}\mathbf{p}_k$ in addition to storage for $\mathbf{x}_k$, $\mathbf{g}_k$, $\mathbf{h}_k$, $\mathbf{p}_k$. Only one matrix-vector multiplication with $\mathbf{K}$ and $\mathbf{M}$ have to be executed per iteration step. The storage requirements of the overall algorithm are thus $(p+8)n$ floating point numbers.

2.2. The block Rayleigh quotient minimization algorithm

Longsine and McCormick [21] suggested several variants for blocking Algorithm 2.2. In this section, we consider a straight-forward generalization of DACG that we call BRQMIN.

The vectors $\mathbf{x}_k$, $\mathbf{g}_k$, $\mathbf{h}_k$, and $\mathbf{p}_k$ of Algorithm 2.2 are replaced by block matrices $\mathbf{X}_k$, $\mathbf{G}_k, \mathbf{H}_k, \mathbf{P}_k \in \mathbb{R}^{n\times q}$, respectively. The normalizations in steps (1) and (8) of Algorithm 2.2 are replaced by Gram-Schmidt orthogonalization with respect to the $\mathbf{M}$-inner product. Step (9) is replaced by the solution of an eigenvalue problem of order $2q$. The eigenvectors associated with the q smallest eigenvalues values determine the next iterate $\mathbf{X}_{k+1}$.

As in the single vector algorithm, there are several possibilities for selecting $\mathbf{B}_k$, the block

generalization of β_k in step (6) of Algorithm 2.2. If the matrix $\mathbf{B}_k$ in

$$\mathbf{P}_k = -\mathbf{H}_k + \mathbf{P}_{k-1}\mathbf{B}_k. \tag{10}$$

is chosen such that $\mathbf{P}_k^T\mathbf{K}\mathbf{P}_{k-1} = \mathbf{O}$, then

$$\mathbf{B}_k = (\mathbf{P}_k^T\mathbf{K}\mathbf{P}_k)^{-1}(\mathbf{P}_k^T\mathbf{K}\mathbf{H}_k) \tag{11}$$

corresponds to (7). Choosing $\mathbf{B}_k$ according to (8) or (9) is also possible [24], but again these expressions are equivalent with (11) only in the context of linear system solving. In numerical experiments we fared best with the choice (11).

The columns of $[\mathbf{X}_k, \mathbf{P}_k]$ (or just of $\mathbf{P}_k$) may become nearly linearly dependent as BRQMIN evolves. Therefore, the matrices $[\mathbf{X}_k, \mathbf{P}_k]^T\mathbf{M}[\mathbf{X}_k, \mathbf{P}_k]$ or $\mathbf{P}_k^T\mathbf{K}\mathbf{P}_k$ on the right hand side of the eigenvalue problem in step (9) of the algorithm and in (11), respectively, may become ill-conditioned. If this ill-conditioning is detected, we replace $\mathbf{P}_k$ with $\mathbf{H}_k$ that effectively restarts the algorithm with the best eigenvector approximations $\mathbf{X}_k$.

Finally, we remark that in our experiments, the first column of $\mathbf{X}_k$ is not always the first to converge and several columns can converge simultaneously. The memory consumption of this algorithm is $(p+8q)n+\mathcal{O}(q^2)$ as we also store the auxiliary block matrices $\mathbf{K}\mathbf{X}_k, \mathbf{M}\mathbf{X}_k, \mathbf{K}\mathbf{P}_k$, and $\mathbf{M}\mathbf{P}_k$.

2.3. The locally-optimal block preconditioned conjugate gradient method

In BRQMIN the Rayleigh quotient is minimized in the $2q$-dimensional subspace generated by the eigenvector approximations $\mathbf{X}_k$ and the search directions $\mathbf{P}_k = -\mathbf{H}_k + \mathbf{P}_{k-1}\mathbf{B}_k$. Instead, Knyazev [18] suggests that the space for the minimization be augmented by the q-dimensional subspace $\mathcal{R}(\mathbf{H}_k)$. The resulting algorithm is deemed 'locally-optimal' because $\rho(\mathbf{x})$ is minimized with respect to all available vectors.

If $\mathbf{d}_j = [\mathbf{d}_{1j}^T, \mathbf{d}_{2j}^T, \mathbf{d}_{3j}^T]^T$, $\mathbf{d}_{ij} \in \mathbb{R}^q$, is the eigenvector corresponding to the j-th eigenvalue of (1) restricted to $\mathcal{R}([\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_{k-1}])$, then the j-th column of $\mathbf{X}_{k+1}$ is the corresponding Ritz vector

$$\mathbf{X}_{k+1}\mathbf{e}_j := [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_{k-1}]\ \mathbf{d}_j = \mathbf{X}_k\mathbf{d}_{1j} + \mathbf{P}_k\mathbf{e}_j, \tag{12}$$

with

$$\mathbf{P}_k\mathbf{e}_j := \mathbf{H}_k\mathbf{d}_{2j} + \mathbf{P}_{k-1}\mathbf{d}_{3j}.$$

Notice that $\mathbf{P}_0$ is an empty matrix such that the eigenvalue problem in step 8 of the locally-optimal block preconditioned conjugate gradient method (LOBPCG), displayed in Algorithm 2.3, has order $2q$ only for $k = 0$. In fact the first step of LOBPCG is equal to the first step of BRQMIN.

The algorithm as proposed by Knyazev [18] was designed to compute just a few eigenpairs and so a memory efficient implementation was not presented. For instance, in addition to $\mathbf{X}_k, \mathbf{R}_k, \mathbf{H}_k, \mathbf{P}_k$, the matrices $\mathbf{M}\mathbf{X}_k, \mathbf{M}\mathbf{H}_k, \mathbf{M}\mathbf{P}_k$ and $\mathbf{K}\mathbf{X}_k, \mathbf{K}\mathbf{H}_k, \mathbf{K}\mathbf{P}_k$ are also stored. The resulting storage needed is prohibitive if more than a handful of eigenpairs are needed.

A more memory efficient implementation results when we proceed as in BRQMIN and iterate with blocks of width q in the space orthogonal to the already computed eigenvectors. The computed eigenvectors are stored in $\mathbf{Q}$ and neither $\mathbf{M}\mathbf{Q}$ nor $\mathbf{K}\mathbf{Q}$ are stored. Hence only storage for $(p + 10q)n + \mathcal{O}(q^2)$ numbers is needed.

ALGORITHM 2.3: **The locally-optimal block preconditioned conjugate gradient method (LOBPCG)**

(1) *Choose random matrix* $\mathbf{X}_0 \in \mathbb{R}^{n\times q}$ *with* $\mathbf{X}_0^T\mathbf{M}\mathbf{X}_0 = \mathbf{I}_q$. *Set* $\mathbf{Q} := [\,]$.
(2) *Compute* $(\mathbf{X}_0^T\mathbf{K}\mathbf{X}_0)\mathbf{S}_0 = \mathbf{S}_0\mathbf{\Theta}_0$, ***(Spectral decomposition)***
where $\mathbf{S}_0^T\mathbf{S}_0 = I_q$, $\mathbf{\Theta}_0 = diag(\vartheta_1, \ldots, \vartheta_q)$, $\vartheta_1 \le \ldots \le \vartheta_q$.
(3) $\mathbf{X}_0 := \mathbf{X}_0\mathbf{S}_0$; $\mathbf{R}_0 := \mathbf{K}\mathbf{X}_0 - \mathbf{M}\mathbf{X}_0\mathbf{\Theta}_0$; $\mathbf{P}_0 := [\,]$; $k := 0$.
(4) while $rank(\mathbf{Q}) < p$ do
(5) *Solve the preconditioned linear system* $\mathbf{N}\mathbf{H}_k = \mathbf{R}_k$
(6) $\mathbf{H}_k := \mathbf{H}_k - \mathbf{Q}(\mathbf{Q}^T\mathbf{M}\mathbf{H}_k)$.
(7) $\widetilde{\mathbf{K}} := [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]^T\mathbf{K}[\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$.
(8) $\widetilde{\mathbf{M}} := [\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]^T\mathbf{M}[\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$.
(9) *Compute* $\widetilde{\mathbf{K}}\widetilde{\mathbf{S}}_k = \widetilde{\mathbf{M}}\widetilde{\mathbf{S}}_k\widetilde{\mathbf{\Theta}}_k$ ***(Spectral decomposition)***
where $\widetilde{\mathbf{S}}_k^T\widetilde{\mathbf{M}}\widetilde{\mathbf{S}}_k = I_{3q}$, $\widetilde{\mathbf{\Theta}}_k = diag(\vartheta_1, \ldots, \vartheta_{3q})$, $\vartheta_1 \le \ldots \le \vartheta_{3q}$.
(10) $\mathbf{S}_k := \widetilde{\mathbf{S}}_k[\mathbf{e}_1, \ldots \mathbf{e}_q]$, $\mathbf{\Theta} := diag(\vartheta_1, \ldots, \vartheta_q)$.
(11) $\mathbf{P}_{k+1} := [\mathbf{H}_k, \mathbf{P}_k]\,\mathbf{S}_{k,2}$; $\mathbf{X}_{k+1} := \mathbf{X}_k\mathbf{S}_{k,1} + \mathbf{P}_{k+1}$.
(12) $\mathbf{R}_{k+1} := \mathbf{K}\mathbf{X}_{k+1} - \mathbf{M}\mathbf{X}_{k+1}\mathbf{\Theta}_k$.
(13) $k := k + 1$.
(14) for $i = 1, \ldots, q$ do ***(Convergence test)***
(15) if $\|\mathbf{R}_k\mathbf{e}_i\| <$ tol then
(16) $\mathbf{Q} := [\mathbf{Q}, \mathbf{X}_k\mathbf{e}_i]$; $\mathbf{X}_k\mathbf{e}_i := \mathbf{t}$, *with* $\mathbf{t}$ *a random vector.*
(17) $\mathbf{M}$-*orthonormalize the columns of* $\mathbf{X}_k$.
(18) end if
(19) end for
(20) end while

In an analogous fashion with BRQMIN, the columns of $[\mathbf{X}_k, \mathbf{H}_k, \mathbf{P}_k]$ may become linearly dependent leading to ill-conditioned matrices $\widetilde{\mathbf{K}}$ and $\widetilde{\mathbf{M}}$ in step (9) of the LOBPCG algorithm. If this is the case we simply restart the iteration with random $\mathbf{X}_k$ orthogonal to the computed eigenvector approximations. More sophisticated restarting procedures that retain $\mathbf{X}_k$ but modified $\mathbf{H}_k$ and/or $\mathbf{P}_k$ were much less stable in the sense that the search space basis again became linearly dependent within a few iterations. Restarting with random $\mathbf{X}_k$ is a rare occurrence and in our experience, has little effect on the overall performance of the algorithm.

3. Algorithms II: Newton iteration-type methods

A second class of algorithms we are going to compare are methods that try to compute the stationary points of the Rayleigh quotient, i.e., the zeros of

$$\mathbf{g}(\mathbf{x}) = \mathbf{grad}\,\rho(\mathbf{x})$$

by means of a Newton iteration. The definition of $\rho(\mathbf{x})$ implies that $\mathbf{g}(\mathbf{x})^T\mathbf{x} = 0$. The Jacobian of $\mathbf{g}$, that is the Hessian [30] of ρ, is

$$\mathbf{H}(\mathbf{x}) = \frac{2}{\mathbf{x}^T\mathbf{M}\mathbf{x}} \left(\mathbf{K} - \rho(\mathbf{x})\mathbf{M} - \mathbf{g}(\mathbf{x})(\mathbf{M}\mathbf{x})^T - (\mathbf{M}\mathbf{x})\mathbf{g}(\mathbf{x})^T\right). \tag{13}$$

Thus, one step of Newton iteration for finding a solution of $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}(\mathbf{x}_k)^{-1}\mathbf{g}_k = \mathbf{x}_k + \mathbf{t}_k, \qquad \mathbf{g_k} := \mathbf{g}(\mathbf{x_k}), \tag{14}$$

where the Newton correction $\mathbf{t}_k$ is the solution of

$$\left(\mathbf{K} - \rho_k\mathbf{M} - \mathbf{g}_k\mathbf{x}_k^T\mathbf{M} - \mathbf{M}\mathbf{x}_k\mathbf{g}_k^T\right)\mathbf{t}_k = -\mathbf{r}_k, \qquad \rho_k := \rho(\mathbf{x}_k), \quad \mathbf{r}_k := \mathbf{r}(\mathbf{x}_k). \tag{15}$$

We remark that the Hessian is always singular when $\mathbf{x}$ is an eigenvector because then $\mathbf{H}(\mathbf{x})\mathbf{x} = -\mathbf{g}(\mathbf{x}) = \mathbf{0}$. If $\mathbf{x}$ is the eigenvector associated with the smallest eigenvalue, then the Hessian is positive semidefinite otherwise for any other eigenvector, the Hessian is indefinite because a non-zero symmetric matrix with the property that $\mathbf{x}^T\mathbf{H}(\mathbf{x})\mathbf{x} = -\mathbf{x}^T\mathbf{g}(\mathbf{x}) = \mathbf{0}$ has at least one negative eigenvalue.

If we require that $\mathbf{x}_k^T\mathbf{M}\mathbf{t}_k = 0$ then (15) is equivalent to

$$\begin{bmatrix} \mathbf{K} - \rho_k\mathbf{M} & \mathbf{M}\mathbf{x}_k \\ \mathbf{x}_k^T\mathbf{M} & 0 \end{bmatrix} \begin{pmatrix} \mathbf{t}_k \\ \tau_k \end{pmatrix} = \begin{pmatrix} -\mathbf{r}_k \\ 0 \end{pmatrix}. \tag{16}$$

This saddle point matrix is nonsingular unless $(\mathbf{x}_k, \rho_k)$ is an eigenpair of (1) with ρ_k a multiple eigenvalue. If $\mathbf{K} - \rho_k\mathbf{M}$ is nonsingular then the solution of (15)-(16) is

$$\mathbf{t}_k = -\mathbf{x}_k + \tau_k(\mathbf{K} - \rho_k\mathbf{M})^{-1}\mathbf{M}\mathbf{x}_k, \tag{17}$$

where

$$\tau_k = -\mathbf{g}_k^T\mathbf{t}_k = -1/(\mathbf{x}_k^T\mathbf{M}(\mathbf{K} - \rho_k\mathbf{M})^{-1}\mathbf{M}\mathbf{x}_k).$$

Substituting (17) into (14) yields

$$\mathbf{x}_{k+1} = \tau_k(\mathbf{K} - \rho_k\mathbf{M})^{-1}\mathbf{M}\mathbf{x}_k \tag{18}$$

that is Rayleigh quotient iteration [26]. Rayleigh quotient iteration has a cubic rate of convergence, i.e., if the sequence $\{\mathbf{x}_k\}_{k=0}^{\infty}$ converges to an eigenvector $\mathbf{u}$ of (1) then the angle $\phi_k := \angle(\mathbf{x}_k, \mathbf{u})$ between $\mathbf{x}_k$ and $\mathbf{u}$ decreases cubically, $|\phi_{k+1}| \le C \cdot |\phi_k|^3$ for large enough k, see [25, 26]. This is more than what can be expected by a Newton iteration.

3.1. The Jacobi-Davidson algorithm (JDSYM)

The convergence behavior of Newton (and of Rayleigh quotient) iteration is unpredictable far away from the solution. Therefore, in the Jacobi-Davidson algorithm [34] the Rayleigh quotient $\rho_k = \rho(\mathbf{x}_k)$ in (15)-(16) is replaced by a *shift* η_k that is held fixed at the beginning of the iteration. Only near convergence is $\eta_k = \rho_k$ in order to exploit the high convergence rate of Rayleigh quotient iteration. Therefore, the Jacobi-Davidson algorithm uses the correction equation

$$(\mathbf{K} - \eta_k\mathbf{M} - \mathbf{g}_k\mathbf{x}_k^T\mathbf{M} - \mathbf{M}\mathbf{x}_k\mathbf{g}_k^T)\mathbf{t}_k = -\mathbf{r}_k, \qquad \mathbf{x}_k^T\mathbf{M}\mathbf{t}_k = 0 \tag{19}$$

which is solved for $\mathbf{t}_k$.

We can, without loss of generality, assume that $\|\mathbf{x}_k\|_{\mathbf{M}} = 1$ and so $\mathbf{g}_k = \mathbf{r}_k$. The constraint $\mathbf{x}_k^T\mathbf{M}\mathbf{t}_k = 0$ implies that $\mathbf{g}_k^T\mathbf{t}_k = \mathbf{x}_k^T(\mathbf{K} - \rho_k\mathbf{M})\mathbf{t}_k = \mathbf{x}_k^T(\mathbf{K} - \eta_k\mathbf{M})\mathbf{t}_k$ whence (19) can be rewritten as

$$\begin{aligned}(\mathbf{K} - \eta_k\mathbf{M})\mathbf{t}_k - \mathbf{M}\mathbf{x}_k\mathbf{x}_k^T(\mathbf{K} - \eta_k\mathbf{M})\mathbf{t}_k = \\ (\mathbf{I} - \mathbf{M}\mathbf{x}_k\mathbf{x}_k^T)(\mathbf{K} - \eta_k\mathbf{M})(\mathbf{I} - \mathbf{x}_k\mathbf{x}_k^T\mathbf{M})\mathbf{t}_k = -\mathbf{r}_k, \qquad \mathbf{x}_k^T\mathbf{M}\mathbf{t}_k = 0.\end{aligned} \tag{20}$$

This form of the correction equation is given by Sleijpen, van der Vorst and coworkers [34, 33, 11], see also [35] for the connection of the Jacobi-Davidson algorithm with Newton's method. Using the relation $\mathbf{r}_k = (\mathbf{K} - \eta_k\mathbf{M})\mathbf{x}_k + (\eta_k - \rho_k)\mathbf{M}\mathbf{x}_k$ implies that the solution of (19)-(20) is

$$\mathbf{t}_k = -\mathbf{x}_k + \tau_k(\mathbf{K} - \eta_k\mathbf{M})^{-1}\mathbf{M}\mathbf{x}_k, \tag{21}$$

where $\tau_k = 1/(\mathbf{x}_k^T\mathbf{M}(\mathbf{K} - \eta_k\mathbf{M})^{-1}\mathbf{M}\mathbf{x}_k)$. If η_k in (19)-(20) is fixed then (14)-(21) becomes

$$\mathbf{x}_{k+1} = \tau_k(\mathbf{K} - \eta_k\mathbf{M})^{-1}\mathbf{M}\mathbf{x}_k \tag{22}$$

which is inverse iteration. If $\eta_k = \rho_k$ then $\mathbf{x}_{k+1}$ is defined by (18) that is Rayleigh quotient iteration.

However, the Jacobi-Davidson algorithm is not just a vector iteration. Following Davidson [9], consecutive corrections $\mathbf{t}_k$ are used to build a search space: In the k-th iteration step of the algorithm the vector $\mathbf{x}_k$ is the Ritz vector corresponding to the Ritz value ρ_k closest to a target σ in the subspace $\mathcal{R}(\mathbf{V}_k)$. The solution $\mathbf{t}_k$ of the correction equation (20) is appended to $\mathbf{V}_k$ to yield $\mathbf{V}_{k+1}$. So, the dimension of the trial space is increased by one in each iteration step. This procedure *accelerates* convergence and is crucial during the beginning of the iteration when the shifts η_k are held fixed for stability reasons.

The memory requirements of the algorithm are constrained by restricting the dimension k of $\mathcal{R}(\mathbf{V}_k)$ to be no larger than input parameter $j_{\max}$. If $k = j_{\max}$ then the iteration is *restarted*, i.e., the trial space is replaced by the $j_{\min}$-dimensional subspace of $\mathcal{R}(\mathbf{V}_k)$ consisting of the Ritz vectors corresponding to the $j_{\min}$ Ritz values closest to the target σ.

The correction equation (20) is approximately solved by a preconditioned conjugate gradient-type method. Of course, this curtails the rate of convergence of the algorithm. However, in the early stages of the iteration, solving the correction equation accurately does not substantially improve the rate of convergence. Therefore, the convergence criterion of the linear system solver is initially loose and tightens as convergence is approached [1].

The preconditioner typically has the form $(\mathbf{I} - \mathbf{M}\widetilde{\mathbf{Q}}_k\widetilde{\mathbf{Q}}_k^T)\mathbf{N}(\mathbf{I} - \widetilde{\mathbf{Q}}_k\widetilde{\mathbf{Q}}_k^T\mathbf{M})$ where $\mathbf{N}$ is an approximation of $\mathbf{K} - \eta_k\mathbf{M}$. Although the coefficient matrix in the correction equation is not positive definite, we nevertheless can choose $\mathbf{N}$ so that the preconditioner is positive definite on $\mathcal{R}(\widetilde{\mathbf{Q}}_k)$. Then we use MINRES [3] as our system solver.

The Jacobi-Davidson algorithm for computing the p smallest eigenvalues of (1) as implemented by Geus [13] is given in Algorithm 3.4. As in the previous algorithms, $\mathbf{Q}$ holds the converged eigenvectors. The matrix $\widetilde{\mathbf{Q}}$ in the correction equation holds the columns of $\mathbf{Q}$ augmented by Ritz vector $\tilde{\mathbf{q}}$. Hence, the correction equation in step (5) of Algorithm 3.4 is solved in the orthogonal complement of the computed eigenvector approximations. The matrix $\mathbf{H}_k$ in steps (7)-(8) is constructed incrementally by appending a row and a column in each iteration step [13]. In steps (14) and (18), respectively $\mathbf{H}_k$ becomes $\mathbf{\Lambda}_k$.

ALGORITHM 3.4: **Jacobi-Davidson algorithm (JDSYM)**

(1) *Choose* $\mathbf{v}_1$ *with* $\|\mathbf{v}_1\|_M = 1$. *Choose target* σ. *Set* $\mathbf{Q} := [\,]$. $k := 1$.
(2) $\mathbf{V}_1 := [\mathbf{v}_1]$; $\tilde{\mathbf{q}} := \mathbf{v}_1$; $\tilde{\rho} := \rho(\tilde{\mathbf{q}})$; $\mathbf{r} := \mathbf{K}\tilde{\mathbf{q}} - \tilde{\rho}\,\mathbf{M}\tilde{\mathbf{q}}$; $\widetilde{\mathbf{Q}} = [\tilde{\mathbf{q}}]$.
(3) while $rank(\mathbf{Q}) < p$ do
(4) *Choose shift:* either $\eta_k := \sigma$ or $\eta_k := \tilde{\rho}$.
(5) *Solve approximately for* $\mathbf{t}$ ***(Correction equation)***
$\{(\mathbf{I} - \mathbf{M}\widetilde{\mathbf{Q}}\widetilde{\mathbf{Q}}^T)(\mathbf{K} - \eta_k\mathbf{M})(\mathbf{I} - \widetilde{\mathbf{Q}}\widetilde{\mathbf{Q}}^T\mathbf{M})\mathbf{t} = -\mathbf{r}, \quad \widetilde{\mathbf{Q}}^T\mathbf{M}\mathbf{t} = 0.\}$
(6) $\mathbf{v}_{k+1} := (\mathbf{I} - \mathbf{V}_k\mathbf{V}_k^T\mathbf{M})\mathbf{t}$; $\mathbf{v}_{k+1} := \mathbf{v}_{k+1}/\|\mathbf{v}_{k+1}\|_M$; $k := k+1$.
(7) $\mathbf{V}_k := [\mathbf{V}_{k-1}, \mathbf{v}_k]$; $\mathbf{H}_k = \mathbf{V}_k^T\mathbf{K}\mathbf{V}_k$. ***(Subspace expansion)***
(8) *Compute* $\mathbf{H}_k\mathbf{S}_k = \mathbf{S}_k\mathbf{\Lambda}_k$ ***(Spectral decomposition)***
where $\mathbf{S}_k^{-1} = \mathbf{S}_k^T$ *and* $\mathbf{\Lambda}_k = diag(\lambda_1^{(k)}, .., \lambda_k^{(k)})$
with $|\lambda_l^{(k)} - \sigma| \le |\lambda_{l+1}^{(k)} - \sigma|$, $l < k$.
(9) repeat ***(Convergence test)***
(10) $\tilde{\rho} := \lambda_1^{(k)}$; $\tilde{\mathbf{q}} := \mathbf{V}_k\mathbf{s}_1$; $\mathbf{r} := \mathbf{K}\tilde{\mathbf{q}} - \tilde{\rho}\,\mathbf{M}\tilde{\mathbf{q}}$.
(11) *found* $:= \|\mathbf{r}\|_2 < \varepsilon$ and $k > 1$;
(12) if *found* then
(13) $\mathbf{Q} := [\mathbf{Q}, \tilde{\mathbf{q}}]$; $\mathbf{V}_{k-1} := \mathbf{V}_k[\mathbf{s}_2, \ldots, \mathbf{s}_k]$;
(14) $\mathbf{\Lambda}_{k-1} := diag(\lambda_2^{(k)}, \ldots, \lambda_k^{(k)})$; $\mathbf{S}_{k-1} := \mathbf{I}_{k-1}$; $k := k-1$;
(15) end if
(16) until *not(found)*
(17) if $k = j_{max}$ then ***(Restart)***
(18) $\mathbf{V}_{j_{min}} := \mathbf{V}_k[\mathbf{s}_1, \ldots, \mathbf{s}_{j_{min}}]$; $k := j_{min}$;
(19) end if
(20) end while

The memory requirement of Algorithm 3.4 is $(p+2j_{\max})n + \mathcal{O}(j_{\max}^2)$ floating point numbers for $\mathbf{Q}$, $\mathbf{V}$, and $\mathbf{MV}$. A few more vectors are needed for the solution of the correction equation by MINRES (or some other Krylov method). Of course, the memory needed for $\mathbf{K}$, $\mathbf{M}$ and the preconditioner have to be included in the total memory requirements.

3.2. The Jacobi-Davidson conjugate gradient algorithm (JDCG)

Notay [23] suggested two modifications to reduce the execution time of the Jacobi-Davidson Algorithm 3.4

- The shift η_k is chosen so that the correction equation in step (6) is positive definite in the relevant subspace $\mathcal{R}(\mathbf{MQ}_{k-1})^\perp$ so that the preconditioned conjugate gradient method replaces MINRES. This reduces the amount of work per iteration step by a factor of two. If $\mathbf{K}$ and $\mathbf{M}$ are both positive definite (or are brought into this form) then η_k can be set to zero initially. As soon as the Rayleigh quotient $\tilde{\rho}$ is close to the desired eigenvalue λ_k then η_k is set to $\tilde{\rho}$; Notay suggests that

$$\|\mathbf{r}_k\|_2 \le \rho_k \quad \text{and} \quad |\rho_k/\rho_{k-1} - 1| \le b,$$

where b is some user provided threshold parameter, as the switching criteria (see [23] for details). Notice, that these criteria are difficult to satisfy if the eigenvalues are clustered.

- Furthermore Notay suggests a sophisticated stopping criterion for the inner iteration. Motivated by the aforementioned observation that the correction equation is not solved accurately in the early stages of the outer iteration, Notay [23] derived a formula for determining the norm of the outer residual as a function of the inner iteration.

These modifications only affect how the correction equations are solved.

4. Algorithms III: The Lanczos iteration

In this section, we briefly overview the implicitly restarted Lanczos algorithm as implemented in ARPACK [20] as applied to (1). ARPACK was selected over others because of the wide availability and high quality implementation.

The Lanczos iteration is a well-known method for calculating a few of the extremal eigenvalues of a large symmetric matrix $\mathbf{A}$. Given an initial starting (unit) vector $\mathbf{q}_1$, the method proceeds by computing the familiar three-term recurrence

$$\beta_j \mathbf{q}_{j+1} = \mathbf{A}\mathbf{q}_j - \alpha_j \mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}, \quad \text{where } \beta_0 \mathbf{q}_0 = \mathbf{0} \text{ and } j \geq 1. \tag{23}$$

The eigenvalues of the symmetric tridiagonal matrix $\mathbf{T}_j$ consisting of α_j's and β_j's on the diagonal and off-diagonals, respectively, are used to estimate those of $\mathbf{A}$.

Equation (23) may be rewritten in matrix form as

$$\mathbf{A}\mathbf{Q}_j = \mathbf{Q}_j \mathbf{T}_j + \beta_j \mathbf{q}_{j+1} \mathbf{e}_j^T, \tag{24}$$

where $\mathbf{Q}_j = [\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j]$, $\mathbf{T}_j$ is the associated tridiagonal matrix, and $\mathbf{e}_j^T$ is the last row of the $j \times j$ identity matrix. We call this a Lanczos reduction of order j. The Lanczos vectors $\mathbf{q}_1, \ldots, \mathbf{q}_j$ provide an orthogonal basis for the Krylov space $\text{span}\{\mathbf{q}_1, \mathbf{A}\mathbf{q}_1, \ldots, \mathbf{A}^{j-1}\mathbf{q}_1\}$, and thus $\mathbf{T}_j$ is the orthogonal projection of $\mathbf{A}$ onto this Krylov space. Notice that the vectors $\mathbf{A}^i\mathbf{q}_1$ are those generated by the power method. So, the Lanczos method is a generalization of the power method in that a sequence of iterates is used to approximate eigenvalues of $\mathbf{A}$.

Unfortunately, round-off errors cause the Lanczos vectors to lose orthogonality [26]. We briefly review two implementations of the Lanczos method that have emerged and their strategy for dealing with the loss of orthogonality.

- Implementations based on selective and partial orthogonalization [14]. These techniques carefully monitor the loss of orthogonality and perform additional orthogonalization steps only when necessary.

- Orthogonalization of each new Lanczos vector against all the vectors generated.

The careful implementation of the first of these approaches is nontrivial. The second approach removes the complication associated with monitoring the loss of orthogonality. The drawbacks are the cost of maintaining orthogonality (on the order of nm^2 floating-point operations for m Lanczos vectors) and storing the Lanczos basis vectors. If m is kept to a moderate size, the cost of maintaining the orthogonality of the Lanczos vectors is not a concern. We emphasize

that the cost of maintaining full orthogonality of the Lanczos basis vectors may represent a minor cost (say, less than 5%) of the total cost in computing the eigenvalues and eigenvectors if m is not large and the cost of computing matrix-vector products $\mathbf{Ax}$ is large.

Suppose that we are able to compute m steps of (23) where m is chosen so that the cost of maintaining the orthogonality among the Lanczos vectors to machine precision is small. Because we are interested in the k largest eigenvalues of $\mathbf{A}$, consider starting another Lanczos iteration such that the first k Lanczos vectors span the same space as the Ritz vectors corresponding to the k largest Ritz values of the previous Lanczos reduction. This *restarting* of the Lanczos method is continued until the k largest eigenvalues of $\mathbf{T}_m$ satisfy the specified tolerance. Implicit restarting [36] is an efficient and numerically stable fashion in which to restart a Lanczos iteration. In particular, implicitly restarting a Lanczos iteration is equivalent to subspace iteration [19]. The reader is referred to ARPACK [20] for further details on an efficient implementation of implicit restarting.

4.1. Shift-and-invert spectral transformation

For computing some of the lowest eigenvalues of the generalized eigenvalue problem (1) it is advisable to transform it to

$$(\mathbf{K}-\sigma\mathbf{M})^{-1}\mathbf{Mx} \equiv \mathbf{Ax} = (\lambda-\sigma)^{-1}\mathbf{x} \equiv \nu\mathbf{x}.$$

This is called a shift-and-invert spectral transformation [2] with shift σ. If $\mathbf{K}$ is positive definite then often $\sigma = 0$ is set. We remark that $\mathbf{A} = (\mathbf{K}-\sigma\mathbf{M})^{-1}\mathbf{M}$ is not symmetric and so the eigenvectors are no longer orthogonal, in general. However, this situation can be remedied by using the inner product $\mathbf{x}^T\mathbf{My}$ defined by the mass matrix $\mathbf{M}$ in the Lanczos procedure. The matrix $(\mathbf{K}-\sigma\mathbf{M})^{-1}\mathbf{M}$ is symmetric with respect to this inner product. The eigenvectors of the matrix stencil $(\mathbf{K};\mathbf{M})$ are also orthogonal with respect to the $\mathbf{M}$-inner product.

The key computational task is the numerical solution of the linear systems

$$(\mathbf{K}-\sigma\mathbf{M})\mathbf{x}_i = \mathbf{Mq}_i \tag{25}$$

necessary at every step of the Lanczos iteration. The vector $\mathbf{x}_i$ is then $\mathbf{M}$-orthogonalized against the columns of $\mathbf{Q}_j$ to obtain $\mathbf{q}_{i+1}$. We employ an inner iteration to approximately solve the above system. We use a preconditioned conjugate gradient iteration as the inner iteration.

Algorithm 4.5 outlines the above procedure. Notice that forming the Lanczos reduction of length m in step (4) amounts to extending an already available Lanczos reduction of length one in the first iteration step and of length j otherwise. Thus, this most time consuming step of the algorithm consists of solving a sequence of $m-j$ systems of equation of the form (25). The Lanczos reductions are computed in the $\mathbf{M}$-orthogonal complement of $\mathcal{R}(\mathbf{Q})$, which holds the already converged eigenvector approximations. This prevents the algorithm from re-computing eigenpairs but is of course a computationally intensive procedure. This procedure may be more time consuming than maintaining the orthogonality among the Lanczos vectors! Step (8) of the algorithm amounts to applying a number of perfect shift QR steps, where the shifts are chosen to be the undesired eigenvalues of $\mathbf{T}_m$.

Algorithm 4.5 requires memory for $\mathbf{Q}$, $\mathbf{Q}_m$, and $\mathbf{MQ}_m$, i.e. $(p+2m)n$ floating point numbers. Furthermore $\mathbf{K}$, $\mathbf{M}$ and the preconditioner have to be stored, and memory for a few additional vectors is required by the conjugate gradient algorithm. For our numerical experiments, the restart size j is set equal to $k+q$ where k is the desired number of eigenvalues and q is the *block*

ALGORITHM 4.5: **Restarted Lanczos Algorithm**

(1) *Choose* $\mathbf{q}_1$ *with* ${\mathbf{q}_1}^T(\mathbf{M}\mathbf{q}_1) = 1$. $\mathbf{Q} = []$.
(3) Iteration
(4) *Form a Lanczos reduction of length* m,
$(\mathbf{K} - \sigma\mathbf{M})^{-1}\mathbf{M}\mathbf{Q}_m = \mathbf{Q}_m\mathbf{T}_m + \beta_m\mathbf{q}_{m+1}\mathbf{e}_m^T$
where $\mathbf{Q}^T\mathbf{M}\mathbf{Q}_m = \mathbf{0}$
(5) *Compute the spectral decomposition* $\mathbf{T}_m\mathbf{S}_m = \mathbf{S}_m\mathbf{\Lambda}_m$
(6) *Extract the converged Ritz vectors from* $\mathbf{Q}_m\mathbf{S}_m$
and append them to $\mathbf{Q}$.
(7) *Implicitly restart and obtain a length* $j \geq 1$ *Lanczos reduction*
$(\mathbf{K} - \sigma\mathbf{M})^{-1}\mathbf{M}\mathbf{Q}_j = \mathbf{Q}_j\mathbf{T}_j + \beta_j\mathbf{q}_{j+1}\mathbf{e}_j^T$,
such that $\mathbf{Q}_j$ *contains the* k *Ritz vectors associated with the smallest* k *eigenvalues of* $\mathbf{T}_m$

size. This allows for a comparison with the block algorithms. We set the maximum dimension for the search space to be $m = k + 2q$.

5. Numerical Experiments

In this section we discuss the numerical experiments used for the comparisons. All the algorithms were implemented in MATLAB using sparse matrix storage and utilities (the only exception is the MATLAB routine EIGS that is an interface to the Fortran library ARPACK [20]). This allowed us the ability to focus on the algorithmic issues and determine general trends. We are making all of our codes available in the public domain so that others, in particular the progenitors of the respective algorithms, may check our results.

The computations were performed using MATLAB Version 6.5 Release 13 on a Pentium IV with a clock rate of 2.4 GHz and a main memory of 1 GB.

We used a MATLAB version of the smoothed aggregation AMG preconditioner described in [38, 39, 29] using a symmetric Gauss-Seidel smoother. The MATLAB preconditioner is a loose implementation of the AMG preconditioner in [17] where the primary difference is in the quality of the agglomeration. For each of the problems we benchmarked, we give convergence rate factors so that the quality of the AMG preconditioner can be ascertained.

The results of the two problems we benchmark are summarized in two tables. The first column gives the block size n_{b} used in the block methods BRQMIN and LOBPCG. The block sizes we tested were $10, 20, 50, 100$ (but less than or equal to the number of desired eigenpairs). The block size n_{b} also is related to the size of the restart space used for JDSYM, JDCG and PIRL; these algorithms were restarted if the dimension of the search space reached $m = n_{\mathrm{eig}} + 2n_{\mathrm{b}}$. The columns indicated by $\mathbf{K}\mathbf{v}$ and $\mathbf{M}\mathbf{v}$ display the number of matrix-vector multiplications executed with the stiffness and mass matrices, respectively. The column $\mathbf{P}^{-1}\mathbf{v}$ gives the number of systems of equations that were solved with the preconditioner $\mathbf{P}$. The column n_{proj} gives an estimate for the number of projections that have been executed in

the respective algorithm to enforce orthogonality within the search space and against the approximate eigenvectors that satisfy the convergence criterion. These are mostly projections of the form $\mathbf{I} - \mathbf{Q}\mathbf{Q}^T\mathbf{M}$. If $\mathbf{Q} \in \mathbb{R}^{n\times q}$ then applying $\mathbf{I} - \mathbf{Q}\mathbf{Q}^T\mathbf{M}$ counts as q projections. The time this latter operation consumes is much higher than the matrix-vector multiplication with $\mathbf{K}$ and $\mathbf{M}$.

All the algorithms were iterated until the individual eigenvector approximations $\mathbf{q}_k$ satisfied $\|\mathbf{K}\mathbf{q}_k - \lambda_k\mathbf{M}\mathbf{q}_k\|_2 \leq 10^{-6}$. This required us to modify the convergence checking subroutine `dsconv.f` in ARPACK to employ the criterion as described [14].† The eigensolver returns if n_{eig} vectors satisfy this criterion. Note, that the matrix residual is larger than the largest individual residual. The columns $\mathcal{O}$, $\mathcal{E}_1$ and $\mathcal{E}_2$ measure the quality of the computed quantities. The column $\mathcal{O}$ gives $\|\mathbf{Q}^T\mathbf{M}\mathbf{Q} - \mathbf{I}_{n_{\text{eig}}}\|_2$ where the matrix $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_{n_{\text{eig}}}]$ contains the n_{eig} computed eigenvector approximations. As can be seen from this column, all algorithms except JDCG give eigenvector approximations that are $\mathbf{M}$-orthogonal to machine precision. The column indicated by $\mathcal{E}_1$ gives $\|\mathbf{Q}^T\mathbf{K}\mathbf{Q} - \mathbf{\Lambda}\|_2$ and the last column indicated by $\mathcal{E}_2$ the matrix residual $\|\mathbf{K}\mathbf{Q} - \mathbf{M}\mathbf{Q}\mathbf{\Lambda}\|_2$.

All algorithms except for PIRL only require a single application of the AMG preconditioner per iteration step; for PIRL, the AMG preconditioner is used in the conjugate gradient iteration for solving systems of equations of the form (25). Each system is solved to an accuracy of 10^{-8} relative to the norm of the initial residual. (Asking for less accuracy destroys the Lanczos relation (24) and leads to unpredictable results). In our computations we always use the same initial vectors. We found that changing the initial vectors can change execution times and operation counts by about 10%.

The next two sections discuss the two problems we benchmarked. Both problems were discretized using a conforming finite element method and use the consistent mass matrix. The first problem is the Laplacian on the unit cube discretized using trilinear elements. The second problem is a plane-stress problem discretized using linear elements on triangles over an unstructured grid. The third section discusses the results.

5.1. Laplace eigenvalue problem

In this section we discuss the solution of the simple model problem

$$-\Delta u(\mathbf{x}) = -\sum_{i=1}^{3} \frac{\partial^2 u}{\partial x_i^2}(\mathbf{x}) = \lambda u(\mathbf{x}), \quad \mathbf{x} \in \Omega = (0,1)^3, \qquad u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega, \tag{26}$$

in three dimensions. The eigenvalues are given by

$$\lambda_{i_1,i_2,i_3} = \omega^2_{i_1,i_2,i_3} = \pi^2(i_1^2 + i_2^2 + i_3^2), \qquad i_1, i_2, i_3 \in \mathbb{N}. \tag{27}$$

Evidently, there are numerous multiple eigenvalues. We are interested in determining a number of the smallest eigenvalues of (26). A finite element discretization with a rectangular n_1-by-n_2-by-n_3 grid using a uniform triangulation with piecewise trilinear elements generates stiffness and mass matrices, $\mathbf{K}$ and $\mathbf{M}$, respectively, of order $n_1n_2n_3$. If we choose $n_i = n$ for all i then

†This required us to build the MATLAB shared library libmwarpack.so using the modified version of the convergence checking subroutine.

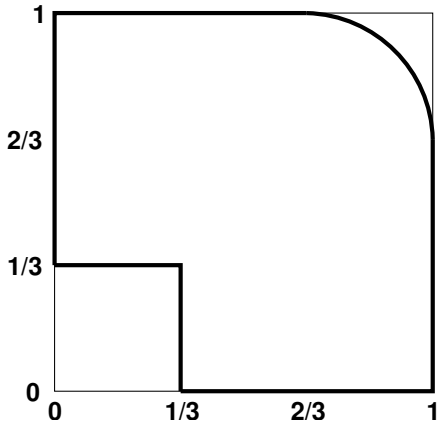

Figure 1. The clamped plate

the discretized problem has numerous multiple eigenvalues. If the n_i differ then most of the multiple eigenvalues separate into clusters of nearby eigenvalues.

We executed experiments with the 3D Laplace problem on a n-by-n-by-n grid where $n = 25$. The problem size is thus $N = n^3 = 15,625$.

In Table I we list the statistics for some of the computations for determining the $n_{\text{eig}} = 10, 50, 100, 200,$ and 500 smallest eigenpairs. The eigenvalue problem has reasonably well separated eigenvalues. The smallest two eigenvalues are $\lambda_1 \approx 29.6$ and $\lambda_2 \approx 59.4$. The largest is $\lambda_{15625} \approx 2.4 \cdot 10^4$. Thus the relative gaps between the smallest eigenvalues are about 10^{-3}. However, many of the eigenvalues are multiple, mostly triple.

In this application the AMG preconditioner had three levels. The three levels had orders 15625, $729 = 9^3$ and 11. The memory consumed by the system matrices on the levels except the finest and by the transfer matrices is about 30% of $\mathbf{K}$. An indication of the quality of the preconditioner is that it reduces the residual norm by a factor 10^6 in five conjugate gradient iterations; this corresponds to a multigrid convergence factor of $\gamma \approx 0.06$ ($\gamma^5 \approx 10^{-6}$).

5.2. Plane stress eigenvalue problem

The following example is taken from the users' guide of the MATLAB PDE toolbox [22, pp.2-38ff].

We consider a 1-by-1 meter steel plate with a right angle inset at the lower left corner and a rounded cut at the upper right corner starting from $(2/3, 1)$ to $(1, 2/3)$, see Fig. 5.2.

The plate is clamped along the inset; the remainder of the boundary remains free. The plate is 1 mm thick. Young's modulus and Poisson's ratio are $E = 196 \cdot 10^3$ MN/m^2 and $\nu = 0.31$, respectively.

We have constructed matrices of order 75168 that arise from an unstructured triangulation using linear elements on triangles. The mass and stiffness matrices are scaled so that $\|\mathbf{K}\|_\infty = \|\mathbf{M}\|_\infty = 1$. The smallest and largest eigenvalues are $\lambda_1 = 9.07 \cdot 10^{-7}$ and $\lambda_{75168} = 7.62$, respectively. The minimum and maximum distance between two eigenvalues is $1.7 \cdot 10^{-7}$ and $5.7 \cdot 10^{-5}$.

solver	n_{b}	t_{exec}	$\mathbf{Kv}$	$\mathbf{Mv}$	$\mathbf{P}^{-1}\mathbf{v}$	n_{proj}	$\mathcal{O}$	$\mathcal{E}_1$	$\mathcal{E}_2$
$n_{\mathrm{eig}} = 10$									
DACG		227	787	794	386	1713	1.2e-14	4.9e-4	9.7e-6
JDSYM	10	116	421	268	191	1340	1.1e-14	1.5e-4	7.9e-6
JDCG	10	115	390	203	194	1505	9.5e-15	4.5e-4	1.1e-5
LOPCG		88	298	319	144	881	3.0e-15	2.9e-4	8.0e-6
BRQMIN	10	42	283	467	240	1022	3.3e-15	3.0e-4	1.0e-5
LOBPCG	10	29	195	304	160	653	2.8e-15	1.4e-5	9.0e-6
PIRL	10	194	689	159	318	104	8.4e-15	1.5e-7	1.4e-7
$n_{\mathrm{eig}} = 50$									
DACG		1476	4724	4792	2324	63624	1.3e-13	1.3e-3	1.3e-5
JDSYM	10	603	2150	1386	971	31386	2.2e-14	3.7e-4	1.0e-5
JDCG	10	677	2239	1171	1116	40611	1.0e-12	1.4e-3	1.2e-5
LOPCG		835	2660	2801	1305	41428	9.4e-15	8.6e-4	1.1e-5
BRQMIN	50	256	1424	2566	1300	30428	4.8e-15	2.4e-4	1.4e-5
	20	243	1748	3282	1600	46130	4.9e-15	7.7e-4	1.9e-5
	10	342	2237	4102	1980	61141	4.8e-15	9.4e-4	1.9e-5
LOBPCG	50	164	913	1556	800	17473	4.4e-15	1.3e-5	1.0e-5
	20	126	888	1657	780	21570	5.6e-15	4.1e-4	1.2e-5
	10	159	1016	1879	870	26188	4.6e-15	6.3e-4	1.2e-5
PIRL	50	724	2626	605	1212	402	1.4e-14	2.7e-7	6.8e-7
	10	440	1573	366	726	240	1.5e-14	9.8e-2	8.4e-2
	20	437	1703	393	786	260	1.1e-14	8.6e-3	2.4e-2
$n_{\mathrm{eig}} = 100$									
BRQMIN	50	618	3310	6343	3100	178123	5.0e-15	9.5e-4	1.9e-5
	20	762	5369	10278	5000	317049	6.0e-15	1.3e-3	2.3e-5
	10	1325	8524	15639	7650	517853	5.0e-15	1.5e-3	2.52e-5
LOBPCG	50	395	1780	3327	1600	87715	5.9e-15	5.1e-4	1.1e-5
	20	358	2449	4702	2200	142663	8.2e-15	7.5e-4	1.2e-5
	10	391	2474	4632	2150	133811	4.8e-15	8.9e-4	1.3e-5
PIRL	50	1084	3913	903	1806	600	1.4e-14	6.8e-7	1.9e-6
	20	870	3133	726	1446	480	1.3e-14	9.7e-1	2.6e-1
	10	803	2873	671	1326	440	1.3e-14	1.6e-1	1.2e-1
$n_{\mathrm{eig}} = 200$									
BRQMIN	50	1993	10347	20316	9900	1273693	7.5e-15	1.9e-3	2.5e-5
	20	2712	18636	35752	17540	2310673	9.3e-15	1.9e-3	2.5e-5
	10	4387	27731	50776	25020	3211950	6.3e-15	1.8e-3	2.7e-5
LOBPCG	50	912	4633	9009	4250	540975	7.5e-15	8.3e-4	1.3e-5
	20	860	5718	11091	5200	651994	9.7e-15	1.1e-3	1.3e-5
	10	997	6138	11502	5390	670075	1.1e-14	1.2e-3	1.4e-5
PIRL	50	2010	7145	1656	3297	1100	2.1e-14	2.5e-3	1.5e-2
	20	1615	5731	1331	2645	880	1.6e-14	8.4e-2	8.1e-2
	10	1593	5595	1312	2582	860	1.7e-14	8.8e-1	2.5e-1
$n_{\mathrm{eig}} = 500$									
BRQMIN	100	7191	32210	63935	31200	10141725	9.5e-15	2.8e-3	2.7e-5
	50	10758	53640	105771	52050	17225488	1.2e-14	2.7e-3	2.8e-5
	10	18073	135810	200978	100000	22231310	1.2e-14	1.5e+3	9.5e00
LOBPCG	100	2878	12413	24379	11500	3686499	1.1e-14	1.3e-3	1.5e-5
	50	3029	14467	28626	13500	4308007	9.1e-15	1.4e-3	1.4e-5
	20	3187	19938	38661	18380	5686325	1.3e-14	1.5e-3	1.6e-5
	10	4207	23958	44517	21290	6649367	1.2e-14	1.6e-3	1.4e-5

Table I. Statistics for some of the MATLAB computations with the 3D Laplace problem using the AMG preconditioner. Execution times (t_{exec}) are given in seconds.

The results listed in Table II were obtained similarly as in the previous section with the AMG preconditioner. The setup of the smoothed aggregation AMG preconditioner provided three levels of order 75168, 7266, and 363, respectively. The AMG preconditioner reduces the residual norm by a factor 10^6 in 22 conjugate gradient iterations; this corresponds to a convergence factor of $\gamma \approx 0.53$ ($\gamma^{22} \approx 10^6$).

The Cholesky factor of the lowest level introduced little fill-in. The stiffness matrix required 12.5 MB of memory in MATLAB's storage scheme. The memory consumed by the coarse level matrices and by the transfer operators is about 90% of storage needed by $\mathbf{K}$; the mass matrix required roughly 50% of the memory of $\mathbf{K}$.

5.3. Discussion

We draw the following immediate conclusions from both tables.

1. The single vector iterations (DACG, JDSYM, JDCG, LOPCG and PIRL) are not competitive with the block methods (BRQMIN and LOBPCG). Of course the former require less memory than the latter. However, if many eigenpairs are computed the difference is not significant.
2. BRQMIN is more sensitive to the block size than LOBPCG. For either algorithm, the number of iteration steps increases as the block size decreases. BRQMIN was more efficient than LOBPCG on the plane stress problem for $n_{\text{eig}} \geq 100$. The execution time and the operation counts of BRQMIN are smaller. Additionally, BRQMIN needs less memory.
3. The execution times increase substantially for block size equal to one hundred when computing $n_{\text{eig}} = 500$ eigenvectors while the operation counts decrease.
4. JDCG and JDSYM are essentially insensitive to the size of the restart and so only one restart size was listed. PIRL needs a large restart size in order to compute accurate results. We remark that while the PIRL results list large residuals, the residual listed is a matrix residual and so one large residual can lead to misleading results. For PIRL, the last several residuals were not fully converged. The apparent failure of the convergence criterion used for PIRL is due to the inner iteration; the criterion adapted from [14] assumes a (sparse) direct solve. PIRL can return residuals that satisfy the convergence criterion but execution time is large.
5. When PIRL converges, the number of applications of the mass, stiffness and preconditioner increases the least amount for PIRL as the number of eigenvectors computed is increased. Our explanation is that the quality of the preconditioner deteriorates as the size of the eigenvalue increases. Recall that the preconditioner remains fixed for all algorithms except for PIRL, which uses a preconditioned inner iteration.
   The number of applications of the preconditioner decreases as the block size increases with LOBPCG and BRQMIN. Apparently, blocking seems to counteract the deterioration of the preconditioner. This is an interesting trend that needs further investigation.

In an attempt to understand why the single vector iterations were so much slower than the block methods, we profiled the codes using MATLAB profiler. The majority of time (at least 50%) was spent in application of the mass and stiffness matrices and preconditioner. The block methods were able to effectively apply these matrices and preconditioner to a block of vectors

solver	n_{b}	t_{exec}	$\mathbf{Kv}$	$\mathbf{Mv}$	$\mathbf{P}^{-1}\mathbf{v}$	n_{proj}	$\mathcal{O}$	$\mathcal{E}_1$	$\mathcal{E}_2$
$n_{\mathrm{eig}} = 10$									
DACG		504	343	349	166	1153	1.1e-12	3.1e-7	1.3e-5
JDSYM	10	629	429	257	200	1967	2.6e-14	3.0e-7	9.9e-6
JDCG	10	442	296	158	147	1483	2.0e-14	3.3e-6	9.8e-6
LOPCG		310	216	240	103	711	6.0e-15	4.2e-8	1.1e-5
BRQMIN	10	120	173	305	140	779	4.6e-15	6.8e-8	1.2e-6
LOBPCG		115	162	282	130	689	9.6e-15	6.7e-8	1.0e-5
PIRL	10	2692	1821	91	895	60	2.1e-14	2.8e-15	1.9e-8
$n_{\mathrm{eig}} = 50$									
DACG		5273	3480	3535	1710	48358	2.2e-11	2.2e-6	1.5e-5
JDSYM	20	3217	2261	1341	1061	42181	3.9e-14	1.2e-6	1.5e-5
JDCG	10	2633	1700	900	849	38265	1.3e-12	4.2e-6	1.5e-5
LOPCG		5116	3346	3490	1648	54300	1.7e-14	1.8e-6	1.6e-5
BRQMIN	50	787	912	1758	800	17765	7.8e-15	3.4e-7	1.5e-5
	20	581	867	1677	760	22092	7.3e-15	9.6e-7	1.7e-5
	10	683	937	1768	800	24925	7.7e-15	1.1e-6	1.8e-5
LOBPCG	50	903	912	1756	800	17865	2.2e-14	3.2e-7	1.3e-5
	20	586	846	1638	740	21427	8.9e-15	9.3e-7	1.5e-5
	10	657	883	1671	750	23317	1.1e-14	9.5e-7	1.4e-5
PIRL	50	12525	8559	451	4204	300	2.4e-14	1.7e-13	2.5e-11
	20	7628	5227	271	2568	180	2.3e-14	9.7e-4	1.6e-2
	10	5974	4099	211	2014	140	3.2e-14	5.5e-3	3.6e-2
$n_{\mathrm{eig}} = 100$									
BRQMIN	50	1504	1781	3529	1600	88322	8.7e-15	1.2e-6	1.9e-5
	20	1309	1904	3740	1700	104719	8.3e-15	1.7e-6	1.9e-5
	10	1836	2364	4461	2050	131629	8.5e-15	1.7e-6	2.3e-5
LOBPCG	50	1575	1781	3529	1600	89255	2.2e-14	1.2e-6	1.5e-5
	20	1660	2364	4576	2100	141512	1.3e-14	1.9e-6	1.6e-5
	10	1900	2461	4627	2120	142505	1.5e-14	2.1e-6	1.6e-5
PIRL	50	16475	11309	601	5554	400	2.2e-14	8.1e-8	1.7e-4
	20	11647	8007	421	3933	280	2.3e-14	6.5e-3	4.5e-2
	10	10178	6903	361	3391	240	2.5e-14	2.1e-2	8.1e-2
$n_{\mathrm{eig}} = 200$									
BRQMIN	50	3187	3665	7365	3350	397793	9.2e-15	2.7e-6	2.1e-5
	20	3351	4649	9148	4220	536640	8.7e-15	2.5e-6	2.6e-5
	10	5359	6985	13067	6160	818483	8.9e-15	2.5e-6	2.7e-5
LOBPCG	50	5090	5649	11111	5150	679115	2.2e-14	2.5e-6	1.7e-5
	20	3839	5172	10082	4660	567726	2.2e-14	3.1e-6	1.7e-5
	10	4941	6332	11851	5530	717345	1.8e-14	3.4e-6	1.7e-5
PIRL	50	24533	16763	901	8231	600	2.4e-14	8.0e-3	5.8e-2
	20	19813	13501	721	6630	480	2.6e-14	4.0e-2	1.2e-1
	10	18320	12407	661	6093	440	2.4e-14	9.5e-2	1.9e-1
$n_{\mathrm{eig}} = 500$									
BRQMIN	100	14997	9080	18379	8400	2520849	9.3e-15	3.8e-6	2.5e-5
	50	12376	12738	25611	11950	3931518	9.5e-15	5.4e-6	3.9e-5
	20	21054	26538	51425	24780	8692422	9.4e-15	5.3e-6	4.3e-5
	10	37283	43651	80369	39220	13709004	9.1e-15	5.2e-6	5.4e-5
LOBPCG	100	23109	13622	26875	12400	3712428	1.7e-14	4.4e-6	1.9e-5
	50	15716	16604	32691	15300	4787033	2.2e-14	5.0e-6	2.0e-5
	20	15152	18902	36619	17280	5545481	2.2e-14	5.2e-6	1.8e-5
	10	17734	20552	38289	18130	5534434	2.0e-14	6.1e-6	2.1e-5

Table II. Statistics for some of the MATLAB computations with the plane stress problem using an AMG preconditioner.

	Laplace			plane-stress		
	K	**M**	$\mathbf{P^{-1}}$	**K**	**M**	$\mathbf{P^{-1}}$
blocked	0.011	0.013	0.14	0.047	0.028	0.74
unblocked	0.014	0.018	0.56	0.060	0.036	2.9
ratio	1.3	1.3	3.9	1.3	1.3	4.0

Table III. Seconds per vector for applying the stiffness, mass matrices and preconditioner on a number of vectors simultaneously (a block at a time) or individually (unblocked).

instead of one and so the number of floating point operations to memory references was much higher.

A least squares fit through the data of the number of matrix vector products against the block size was computed in order to access the benefit of block algorithms. The results are listed in Table III. The first and second rows list seconds per vector for applying the stiffness, mass matrices and preconditioner on a number of vectors simultaneously (a block at a time) or individually (unblocked). The last row gives the ratios of blocked to unblocked operations. These times have been obtained with block sizes varying between one and one hundred. The data implies that blocking reduces the execution times by a factor two for matrix-vector multiplication and by a factor of four when applying the preconditioner. The timings show that applying the preconditioner is 50–100 times more expensive than a matrix-vector multiplication. We believe this finding will also be the case when using a compiled computer language because of the deep memory hierarchies now prevalent.

Do these results change if the multiple eigenvalues of problem one become clustered eigenvalues (accomplished by varying the number of grid points in each space dimension)? With smaller two-dimensional Laplace test problems we observed that the algorithms were not sensitive to small changes in the eigenvalues except for DACG and LOPCG. The execution times of these two algorithms increased by 50% to 100% when the eigenvalues are in clusters of size 10^{-4} as opposed to the case when there are actual multiplicities. We remark that DACG and LOPCG are the two true single vector iterations.

As an experiment we ran both problems on all the algorithms using the 'perfect' preconditioner—the Cholesky factors of the stiffness matrix. In this situation PIRL was the most efficient solution when computing 100 and 500 eigenvectors; the latter case saw PIRL twice as fast as LOBPCG, which itself was faster than the numbers given in Table II. This finding conforms with similar comparisons made in [1] that shift-and-invert Lanczos or Arnoldi methods are the most efficient solvers when $\mathbf{K} - \sigma\mathbf{M}$ can be factored.

6. Conclusion

The goal of our report was to compare a number of algorithms for computing a large number of eigenvectors of the generalized eigenvalue problem arising from a modal analysis of elastic structures using preconditioned iterative methods. After a review of various iteration schemes, a substantial amount of experiments were run on two problems using an AMG preconditioner.

Our overall conclusion is that the block algorithms, in particular LOBPCG, are the most efficient. This statement holds under the condition that matrix-vector multiplications and,

most significantly, the application of the preconditioner *are* applied in a blocked fashion and the block size is selected as large as possible.

Finally, if the stiffness matrix $\mathbf{K}$ can be factored, then the shift-and-invert Lanczos algorithm (for instance, as implemented in PIRL) becomes the algorithm of choice.

Acknowledgments

We thank Roman Geus, ETH Zurich, and Yvan Notay, Free University of Brussels, for providing us with MATLAB codes of their Jacobi-Davidson implementations; Ray Tuminaro, Sandia National Laboratories, for his MATLAB implementation of the smoothed aggregation multilevel algorithm ML and Andrew Knyazev, University of Colorado at Denver, for several helpful discussions.

REFERENCES

1. P. Arbenz and R. Geus, *A comparison of solvers for large eigenvalue problems originating from Maxwell's equations*, Numer. Lin. Alg. Appl., 6 (1999), pp. 3–16.
2. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.
3. R. Barret, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1994. (Available from Netlib at URL `http://www.netlib.org/templates/index.html`).
4. L. Bergamaschi, G. Gambolati, and G. Pini, *Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem*, Numer. Linear Algebra Appl., 4 (1997), pp. 69–84.
5. L. Bergamaschi, G. Pini, and F. Sartoretto, *Parallel preconditioning of a sparse eigensolver*, Parallel Comput., 27 (2001), pp. 963–976.
6. L. Bergamaschi and M. Putti, *Numerical comparison of iterative eigensolvers for large sparse symmetric postive definite matrices*, Comput. Methods Appl. Mech. Engrg., 191 (2002), pp. 5233–5247.
7. A. Björck, *Numerics of Gram–Schmidt orthogonalization*, Linear Algebra Appl., 197/198 (1994), pp. 297–316.
8. W. W. Bradbury and R. Fletcher, *New iterative methods for solution of the eigenproblem*, Numer. Math., 9 (1966), pp. 259–267.
9. E. R. Davidson, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices*, J. Comput. Phys., 17 (1975), pp. 87–94.
10. Y. T. Feng and D. R. J. Owen, *Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems*, Internat. J. Numer. Methods Engrg., 39 (1996), pp. 2209–2229.
11. D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, *Jacobi-Davidson style QR and QZ algorithms for the partial reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.
12. G. Gambolati, F. Sartoretto, and P. Florian, *An orthogonal accelerated deflation technique for large symmetric eigenvalue problem*, Comput. Methods Appl. Mech. Engrg., 94 (1992), pp. 13–23.
13. R. Geus, *The Jacobi-Davidson algorithm for solving large sparse symmetric eigenvalue problems*, PhD Thesis No. 14734, ETH Zurich, 2002.
14. R. Grimes, J. G. Lewis, and H. Simon, *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 228–272.
15. M. R. Hestenes and W. Karush, *A method of gradients for the calculation of the characteristic roots and vectors of a real symmetric matrix*, Journal of Research of the National Bureau of Standards, 47 (1951), pp. 45–61.
16. W. Hoffmann, *Iterative algorithms for Gram-Schmidt orthogonalization*, Computing, 41 (1989), pp. 335–348.
17. J. Hu, C. Tong, and R. S. Tuminaro, *ML 2.0 smoothed aggregation user's guide*, Tech. Report SAND2001-8028, Sandia National Laboratories, December 2000.

18. A. V. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.
19. R. B. Lehoucq, *Implicitly restarted Arnoldi methods and subspace iteration*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 551–562.
20. R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems by Implicitely Restarted Arnoldi Methods*, SIAM, Philadelphia, PA, 1998. (The software and this manual are available at URL `http://www.caam.rice.edu/software/ARPACK/`).
21. D. E. Longsine and S. F. McCormick, *Simultaneous Rayleigh–quotient minimization methods for* $Ax = \lambda Bx$, Linear Algebra Appl., 34 (1980), pp. 195–234.
22. The MathWorks, *Partial Differential Equation Toolbox User's Guide, Version 1.0.4*, July 2002. Available online at URL `http://www.mathworks.com/products/pde/`.
23. Y. Notay, *Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem*, Numer. Lin. Alg. Appl., 9 (2002), pp. 21–44.
24. D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.
25. A. M. Ostrowski, *On the convergence of the Rayleigh quotient iteration for the computation of the characteristic roots and vectors. I*, Arch. Rational Mech. Anal., 1 (1958), pp. 233–241.
26. B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980. (Republished by SIAM, Philadelphia, 1998.).
27. A. Perdon and G. Gambolati, *Extreme eigenvalues of large sparse matrices by Rayleigh quotient and modified conjugate gradients*, Comput. Methods Appl. Mech. Engrg., 56 (1986), pp. 125–156.
28. E. Polak, *Computational Methods in Optimization: A Unified Approach*, Academic Press, New York, 1971.
29. G. Poole, Y.-C. Liu, and J. Mandel, *Advancing analysis capabilities in ANSYS through solver technology*. Paper presented at the Tenth Copper Mountain Conference on Multigrid Methods, April/October 2001. Available at URL `http://www-math.cudenver.edu/~jmandel/papers/amg2001.pdf`. To appear in Electronic Transactions on Numerical Analysis.
30. A. Ruhe, *Computation of eigenvalues and vectors*, in Sparse Matrix Techniques, V. A. Barker, ed., Lecture Notes in Mathematics 572, Berlin, 1977, Springer-Verlag, pp. 130–184.
31. H. R. Schwarz, *Rayleigh–Quotient–Minimierung mit Vorkonditionierung*, in Numerical Methods of Approximation Theory, Vol. 8, L. Collatz, G. Meinardus, and G. Nürnberger, eds., vol. 81 of International Series of Numerical Mathematics (ISNM), Basel, 1987, Birkhäuser, pp. 229–45.
32. ———, *Methode der finiten Elemente*, Teubner, Stuttgart, 3rd ed., 1991.
33. G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst, *Jacobi-Davidson type methods for generalized eigenproblems and polynomial eigenproblems*, BIT, 36 (1996), pp. 595–633.
34. G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi-Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.
35. G. L. G. Sleijpen and H. A. van der Vorst, *The Jacobi-Davidson method for eigenvalue problems and its relation with accelerated inexact newton scheme*, in Second IMACS International Symposium on Iterative Methods in Linear Algebra, S. D. Margenov and P. S. Vassilevski, eds., vol. 3 of IMACS Series in Computational and Applied Mathematics, New Brunswick, NJ, 1996, IMACS, pp. 377–389.
36. D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.
37. K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309.
38. P. Vaněk, J. Mandel, and M. Brezina, *Algebraic multigrid based on smoothed aggregation for second and fourth order problems*, Computing, 56 (1996), pp. 179–196.
39. P. Vaněk, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numer. Math., 88 (2001), pp. 559–579.